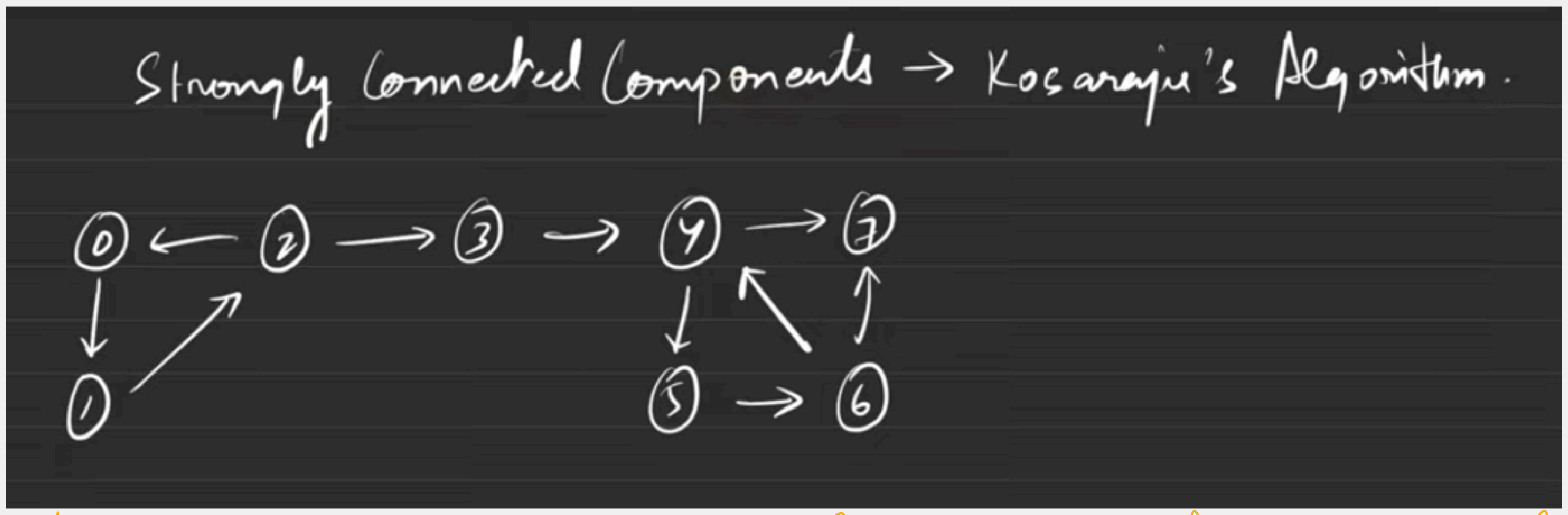
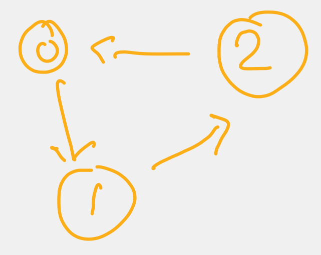
Strongly connected graphs are only valid for directed graphs.



Strongly connected component means for every edge if we can go from node 1 to node 2 we should be able to go from node 2 to node 1 like if we consider below component it is a strongly connected component.

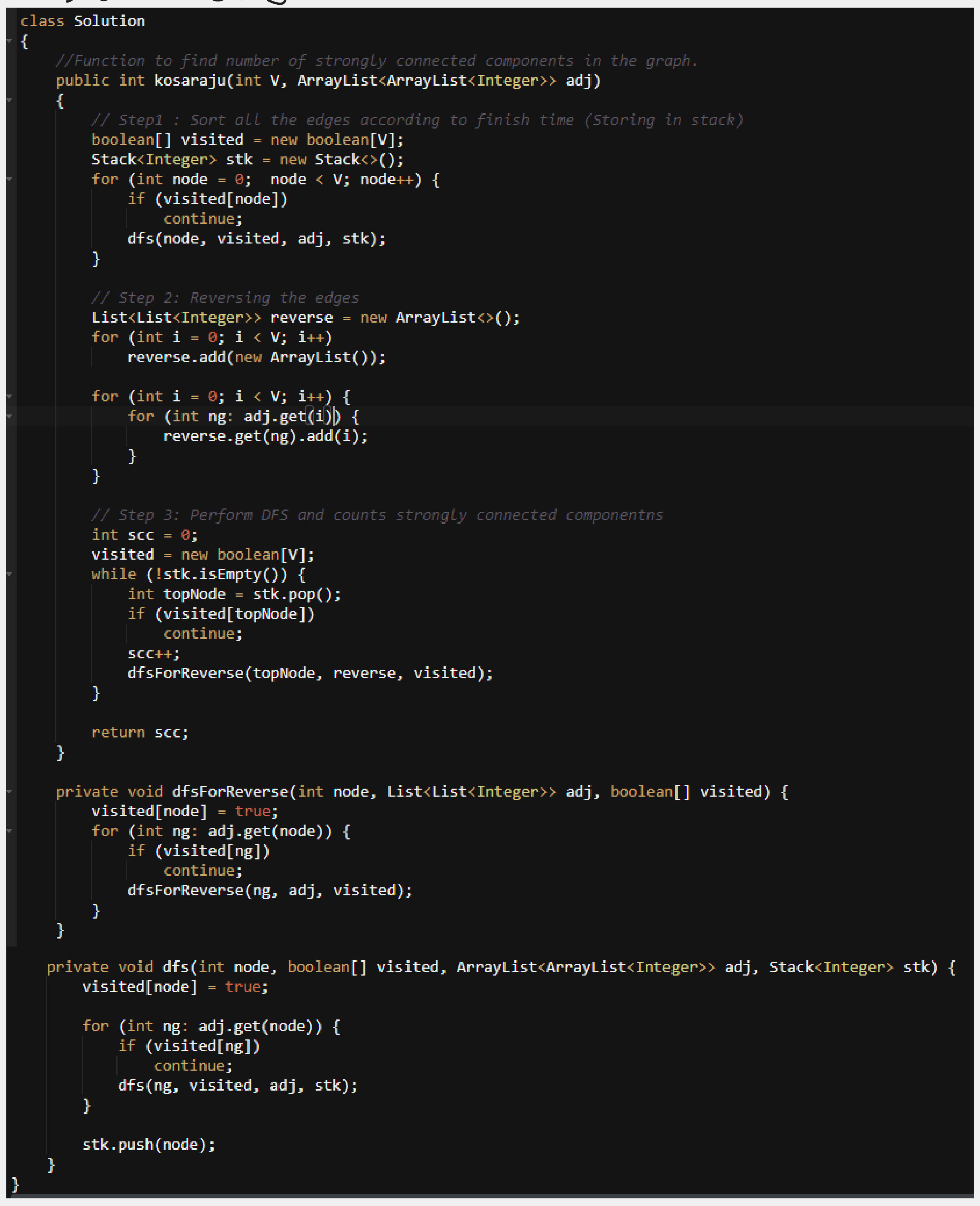


Below are strongly connected components from above example

(0, 1, 2), (3), (4, 5, 6), (7)

The thought process behind this algorithm is that if we reverse the edges we won't be able to visit the other strongly connected components.

1) Sort all the edges according to finishing time.
2) Reverse the graph
3) Do a DFS



```java
class Solution
{
    //Function to find number of strongly connected components in the graph.
    public int kosaraju(int V, ArrayList<ArrayList<Integer>> adj)
    {
        // Step1 : Sort all the edges according to finish time (Storing in stack)
        boolean[] visited = new boolean[V];
        Stack<Integer> stk = new Stack<>();
        for (int node = 0;  node < V; node++) {
            if (visited[node])
                continue;
            dfs(node, visited, adj, stk);
        }

        // Step 2: Reversing the edges
        List<List<Integer>> reverse = new ArrayList<>();
        for (int i = 0; i < V; i++)
            reverse.add(new ArrayList());

        for (int i = 0; i < V; i++) {
            for (int ng: adj.get(i)) {
                reverse.get(ng).add(i);
            }
        }

        // Step 3: Perform DFS and counts strongly connected componentns
        int scc = 0;
        visited = new boolean[V];
        while (!stk.isEmpty()) {
            int topNode = stk.pop();
            if (visited[topNode])
                continue;
            scc++;
            dfsForReverse(topNode, reverse, visited);
        }

        return scc;
    }

    private void dfsForReverse(int node, List<List<Integer>> adj, boolean[] visited) {
        visited[node] = true;
        for (int ng: adj.get(node)) {
            if (visited[ng])
                continue;
            dfsForReverse(ng, adj, visited);
        }
    }

   private void dfs(int node, boolean[] visited, ArrayList<ArrayList<Integer>> adj, Stack<Integer> stk) {
       visited[node] = true;

       for (int ng: adj.get(node)) {
           if (visited[ng])
               continue;
           dfs(ng, visited, adj, stk);
       }

       stk.push(node);
   }
}
```

TC - O(V+E)
SC - O(V) + O(V+E) ← This is for reverse graph.